PROGETTAZIONE E REALIZZAZIONE DI UN'APPLICAZIONE PER LA RACCOLTA E IL CAMPIONAMENTO DI PAGINE WEB

FLAVIO MARCATO

Relatore: Prof. Luca Pretto

Correlatore: Prof. Enoch Peserico

Facoltà di Ingegneria

Corso di laurea di Ingegneria Informatica

Anno accademico 2009-2010

30 Settembre 2010

Flavio Marcato : *Progettazione e realizzazione di un'applicazione per la raccolta e il campionamento di pagine web* , Relazione di laurea , © 30 Settembre 2010

A mio padre e mia madre

SOMMARIO

I primi sistemi di Information Retrieval lavoravano su collezioni di qualità omogenea come documenti giuridici e articoli medici. Con l'avvento del web, le tecniche tradizionali di reperimento dell'informazione sono risultate poco efficaci in quanto incapaci di distinguere la qualità dei documenti; di qui la necessità di ideare algoritmi in grado di selezionare le pagine web in base sia alla rilevanza che alla qualità. Tra questi algoritmi, un posto di rilievo hanno assunto quelli di *link analysis*, che cercano di inferire la qualità delle pagine web dalla struttura topologica del grafo associato al web. Il lavoro descritto in questa relazione è stato svolto all'interno di un progetto che ha lo scopo di valutare l'effettiva efficacia di tali algoritmi.

Il nostro lavoro è consistito nello sviluppo di un'applicazione web che, data un'opportuna popolazione di pagine web, metterà a disposizione una serie di funzionalità mirate alla raccolta di giudizi sulla qualità delle pagine stesse. Il software citato esegue una pre-elaborazione dei risultati restituiti dai motori di ricerca e a tal proposito sono stati sviluppati tre moduli: Interrogatore, che si preoccuperà di estrapolare gli URL dai risultati; Campionatore che, data una teoria euristica ragionevole, filtrerà i risultati restituiti dall'Interrogatore e infine Downloader che si occuperà di memorizzare le pagine su disco.

INDICE

ELENCO DELLE FIGURE

ELENCO DELLE TABELLE

1 INTRODUZIONE

1.1 GLI ALGORITMI DI LINK ANANLYSIS SONO DAVVERO EFFICACI?

Così come già affermato nel 2004 in [1, Pew Internet Survey], il più delle volte chiunque abbia intenzione di effettuare una ricerca lo fa sul web. La quantità di informazioni presente in rete, come conseguenza del progressivo aumento della domanda degli utenti, è cresciuta di anno in anno ed è attualmente nell'ordine dei milioni di Terabyte (stimato dal CEO di Google Eric Schmidt). Una quantità di dati simile ha rappresentato e rappresenta il giusto "terreno" per lo sviluppo dei motori di ricerca ovvero applicazioni in grado di recuperare pagine web a partire da un'interrogazione di un utente. Allo scopo di reperire pagine di qualità sono stati proposti algoritmi di *link analysis* come [2, PAGERANK], [3, HITS] o [4, SALSA]. Nel presente progetto ci proponiamo di valutare sperimentalmente l'effettiva efficacia di questi algoritmi.

1.2 LO SCOPO DEL PROGETTO

Al fine di testare l'effettiva efficacia degli algoritmi di LA è nato a Padova un progetto di ricerca presso il Dipartimento di Ingegneria dell'Informatazione. Per efficacia si intende l'effettiva corrispondenza tra l'ordinamento proposto da un motore di ricerca e la qualità delle pagine web corrispondenti. L'idea è quella di raccogliere in primo luogo più informazioni possibili su un sottoinsieme adeguato del web e successivamente somministrare ad un'utenza predeterminata una popolazione di pagine con le medesime caratteristiche di tutto il web stesso. Una volta ottenuti i voti desiderati si provvederà a confrontare i giudizi "umani" di rilevanza e qualità con i risultati degli algoritmi di LA.

Il team composto da M. Compagnin, R. Compostella, F. Marcato e G. Pengo si è occupato di definire il dominio di lavoro e del perfezionamento dell'applicazione prototipo messa a disposizione da due studenti di Ingegneria. A livello di gruppo sono stati portati a termine i seguenti compiti:

1. Studio di fattibilità del *crawling* del web italiano — *gruppo Compagnin-Pengo*
2. Implementazione e avvio di un *crawler* affidabile
3. Perfezionamento della *web application* — *gruppo Compostella-Marcato*
4. Studio di una teoria di campionamento applicabile
5. Costituzione della popolazione di pagine da valutare

Le scelte preliminari più di rilievo riguardano la definizione del dominio di lavoro per quanto riguarda il *crawling* e il motore di ricerca.

Il crawling verrà effettuato solo sul web italiano in quanto:

- è infattibile analizzare il web globale a causa della sua vastità
- il web italiano è ragionevolmente isolato in termini di link uscenti ed entranti
- le pagine in italiano sono più facilmente valutabili da un'utenza italiana

Il motore di ricerca predefinito:

- è uno solo poichè combinare risultati di più motori di ricerca è eccessivamente oneroso e non porta benefici tangibili
- è Google Search in quanto come rilevato da [5, Hitwise] da solo copre il 71.27% delle ricerche su tutto il web

Nella seguente relazione si presenteranno brevemente l'interfaccia web e la base di dati mentre verrà descritto nel dettaglio lo sviluppo dell'applicazione di *parser* e campionamento.

2 IL PROGETTO SULLA QUALITÀ DELLE PAGINE WEB

2.1 LA WEB APPLICATION

Una parte importante del progetto sulla qualità consiste in una *web application* realizzata in php. Lo scopo dell'applicazione è quello di guidare l'utente attraverso una pagina di registrazione/login (necessaria per fini statistici), per poi introdurlo nella sezione "interrogazioni" dove potrà scegliere una o più *query* tra le 10 proposte. Tale lista di interrogazioni include un insieme di siti di qualità mista in modo che risulti fattibile un riscontro tra le preferenze di utenti umani e quelle di un motore di ricerca automatizzato. Infine sarà possibile votare le pagine proposte mediante una serie di indici rappresentanti rilevanza e qualità.

2.1.1 *Aspetti salienti*

Nella versione stabile il form di login accetterà come *username* un indirizzo di posta elettronica valido più una password definita dall'utente. La scelta legata allo *username* è dovuta al fatto che la mail è unica e se attiva non si dimentica. Ad ogni modo è presente anche una funzionalità di recupero credenziali sempre nella medesima pagina.

Una volta effettuata la login la sezione saliente dell'applicazione web è la pagina di votazioni. Strutturalmente questa pagina è caratterizzata da iframe: l'utente finale avrà a disposizione tutti gli indici di cui ha bisogno sulla destra e la pagina da votare sulla sinistra. In ordine di esecuzione il flusso di operazioni è il seguente:

1. Visione della pagina proposta
2. Giudizio di rilevanza attraverso un voto che va da 1 a 10
3. Se il voto di rilevanza supera il valore 5 allora saranno sbloccati automaticamente tutti gli altri indici
4. Valutazione della pagina attraverso tutti gli indici proposti

In queste fasi una base di dati[1] viene richiamata sia per fornire le pagine da valutare, che per memorizzare i voti assegnati.

È ancora da decidere il numero di pagine che l'utente sarà chiamato a votare, è tuttavia stimato che questo numero si aggiri tra 5 e 15.

1 Per informazioni più dettagliate si rimanda il lettore all'appendice dove allegata troverà tutta la documentazione relativa.

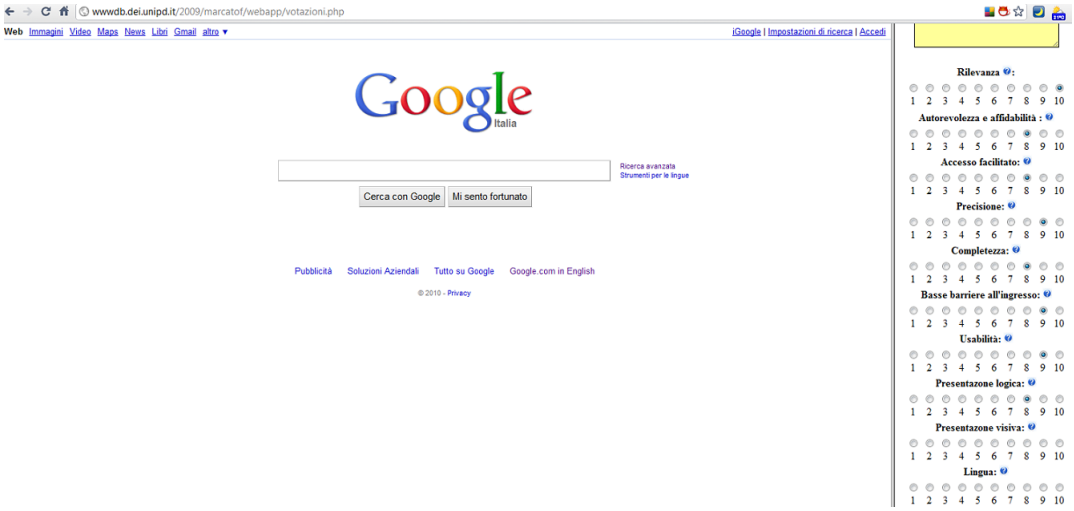

Figura 1: Screen shot votazioni

Una volta che l'utente ha terminato di votare potrà scegliere se proseguire con un'altra interrogazione oppure semplicemente uscire dall'applicazione.

2.2 PROBLEMI DA RISOLVERE

In realtà la *web application* presenta solo un'interfaccia tra l'utente e il cuore dell'intero *software*. Prima di proporre le pagine la popolazione di URL, memorizzata nella base di dati, deve essere preprocessata. Per ogni interrogazione un qualunque motore di ricerca proporrebbe migliaia di risultati ed è impensabile darle in pasto tutte ad un'utenza limitata. Per questo è necessario predisporre un modulo di campionamento che "prepari il terreno" all'applicazione web fornendo tutte e sole le pagine necessarie. Si studierà poi il funzionamento dei motori di ricerca al fine di poter valutare la loro efficacia e infine predisporre un metodo per gestire i voti.

3

L'APPLICAZIONE

L'applicazione web ha una struttura modulare. Ogni modulo riveste una funzione specifica che di seguito verrà esposta nel dettaglio. La scelta di modularità è stata fatta considerando che software con tale struttura hanno l'attitudine ad essere facili da modificare, circoscrivono la manifestazione di errori e sono più agevoli da rappresentare. Di seguito si propone un primo schema generale riguardante tutta l'applicazione nel suo complesso:

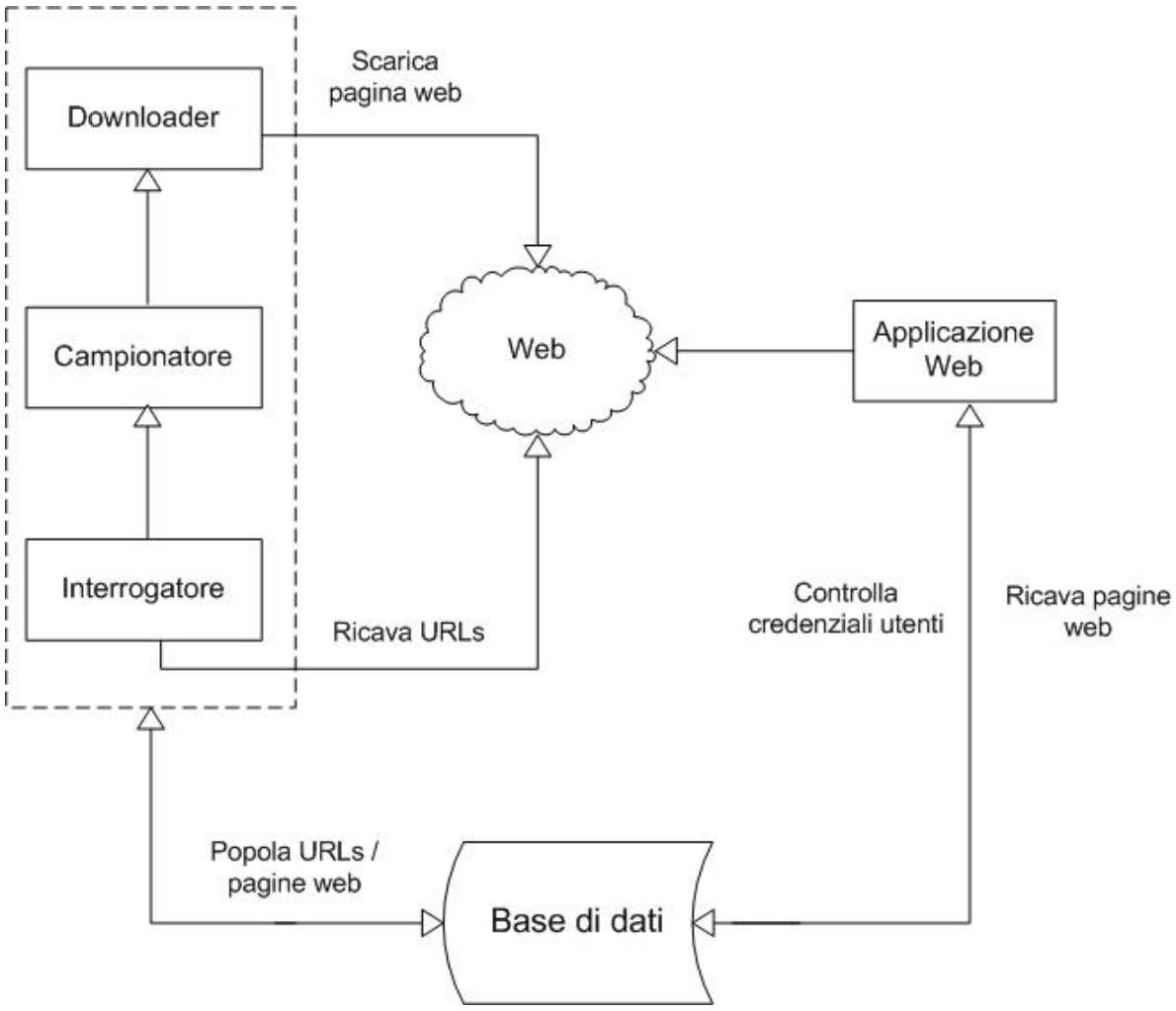

Figura 2: Diagramma Applicazione

Premessa: uno dei requisiti posti a priori riguarda la portabilità del software. Le varie scelte progettuali sono state fatte partendo dal presupposto che tutti gli strumenti messi a disposizione dal dipartimento garantiscano che il requisito citato sia soddisfatto. Per esempio, l'uso del linguaggio *bash* è stato considerato come un valido strumento di sviluppo anche se in realtà limita la portabilità del software a sistemi operativi Linux o affini.

3.1 LA BASE DI DATI

In questa sezione si descriverà sintenticamente il modulo che si preoccupa di gestire i dati. Benchè si tratti in realtà del nucleo dell'applicazione si ricorda che il *focus* del presente elaborato riguarda la parte di software adibita al campionamento.

Inizialmente si è scelto di utilizzare un dbms relazionale quale PostgreSQL in quanto già installato sulle macchine messe a disposizione dal dipartimento e data la buona familiarità degli sviluppatori in questo ambiente. La base di dati ha una duplice utilità: da una parte è fondamentale per la memorizzazione delle informazioni riguardanti gli algoritmi di ricerca, le informazioni sugli utenti, le loro preferenze sulla rilevanza e le votazioni sui vari indici di qualità; dall'altra svolge un'importante ruolo nel supporto delle funzionalità della *web application* in quanto memorizza tutte le informazioni necessarie per le soglie di controllo e tutte le informazioni riguardanti la popolazione di pagine web pronta per essere votata.

Il lavoro è partito da un prototipo elaborato da due studenti di Ingegneria Informatica. La base di dati è stata analizzata e confrontata con le esigenze del progetto e a tal proposito si evidenziano due cambiamenti principali:

1. sono state apportate modifiche a livello concettuale e quindi a livello logico, in particolare:
   a) all'entità Utente semplificando le credenziali richieste al *login* (mail + password) in modo da rendere più agevole la registrazione e di alleggerire l'occupazione in memoria da dati sensibili.
   b) si è modificata l'entità Url a cui si è aggiunto l'attributo "path" che rappresenta il percorso della pagina web su disco.
2. è stata ristrutturata la documentazione correggendone la forma e standardizzandola nel formato.

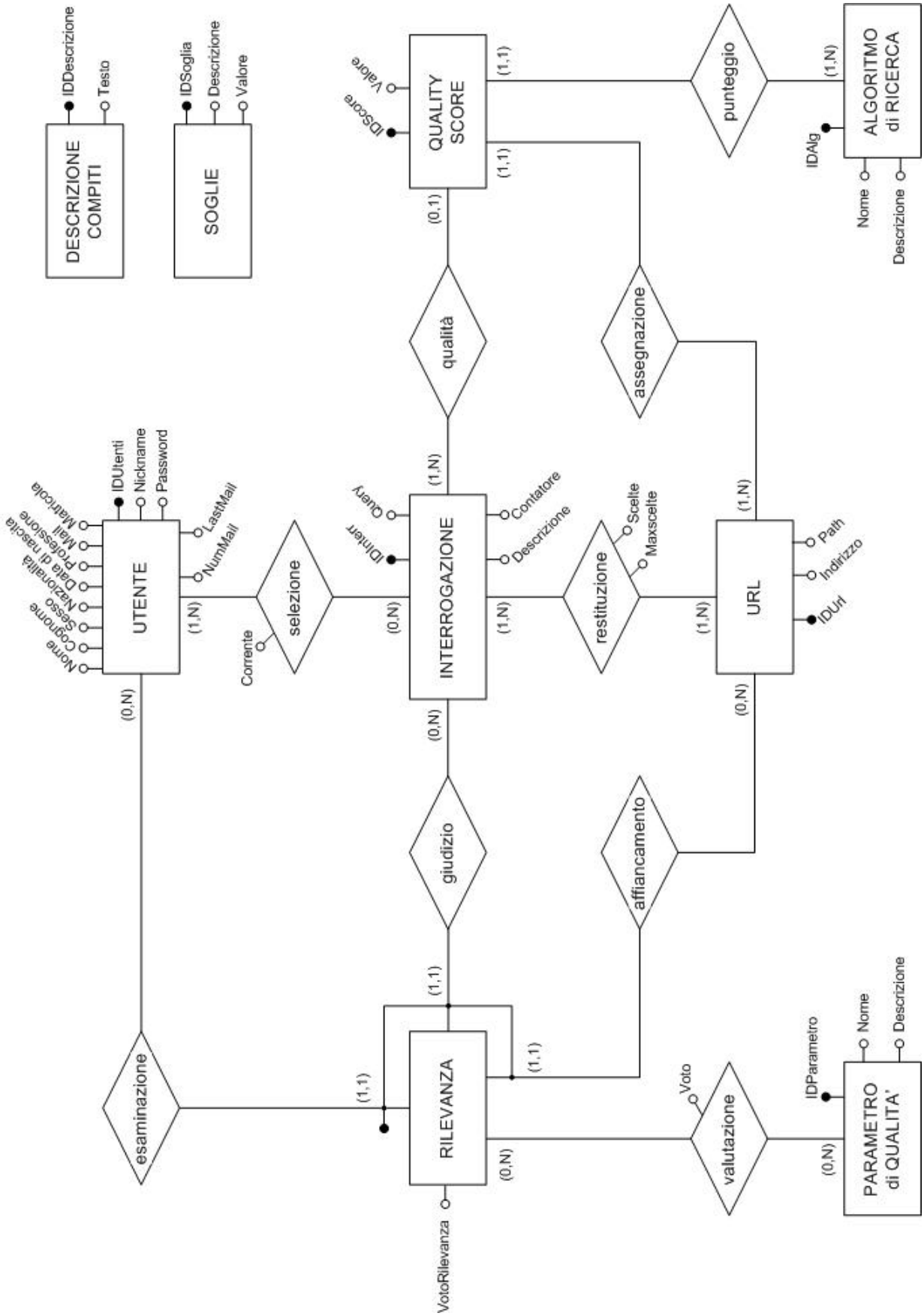

Figura 3: Schema entità-associazione

3.2 INTRODUZIONE AL MODULO CAMPIONAMENTO

Il modulo campionamento è stato sviluppato *ex-novo* nel preciso intento di risolvere il seguente problema: ricavare una popolazione di pagine web valutabile in maniera automatica e seguendo una teoria di campionamento adeguata. In altre parole l'applicazione mira a rendere disponibile una serie di pagine web con qualità stimata mista in modo che l'utente abbia probabilità p di trovare una pagina rilevante e la restante 1-p di trovare una pagina poco rilevante.

Lo sviluppo di tale questione fa emergere tre ulteriori problemi: il primo riguarda l'interrogazione al motore di ricerca e l'estrazione degli url relativi alle pagine indice[1], il secondo è connesso alla necessità di implementare una teoria di campionamento su tutti gli url estratti, il terzo attiene la necessità di mantenere omogenea la popolazione anche a distanza di tempo.

Come motore di ricerca predefinito si è scelto di utilizzare Google.

3.3 FUNZIONALITÀ I: INTERROGAZIONE AL MOTORE DI RICERCA

In questa sezione si esporranno sinteticamente le ipotesi di lavoro storicamente prese in considerazione e si discuteranno le successive scelte progettuali che porteranno alla soluzione finale proposta ed integrata nell'applicazione principale.

La prima scelta significativa che si è dovuta compiere è stata quella relativa al linguaggio o i linguaggi di programmazione da utilizzarsi nello sviluppo. Alla luce dei requisiti fondamentali alla base dell'intera applicazione (portabilità e usabilità) tale scelta è stata effettuata tra le tre principali correnti di pensiero qui di seguito esposte:

1. Scrittura di uno script in *php* da utilizzarsi parallelamente alla *web application*
2. Scrittura di uno script in linguaggio *bash* indipendente dalla *web application*
3. Scrittura di uno script indipendente dalla *web application* con una prima sezione in linguaggio *bash* e una seconda sezione in linguaggio Python.

3.3.1 *La scelta*

La prima opzione a causa dell'eccessivo ritardo che introduceva all'intera applicazione è stata prontamente scartata mentre la seconda e la terza sono risultate invece entrambe fattibili e rispettose di tutti i requisiti posti a monte. La migliore tra tutte è risultata essere la terza opzione per i motivi di seguito elencati: il vantaggio principale riguarda la potenza di Python, un linguaggio molto versatile e progettato appositamente per lavorare in autonomia o in concomitanza ad altri linguaggi. La prima domanda delle FAQ sul sito ufficiale [6, `http://python.org/`]

1 Con pagina indice si intende la pagina che raccoglie tutti gli url risultato della query, in altre parole è la pagina che appare subito dopo aver cliccato su Google Search.

è "Cos'è Python?" "Python è un linguaggio di programmazione interpretato, interattivo e orientato agli oggetti. Incorpora al suo interno moduli, eccezioni, tipizzazione dinamica, tipi di dato di altissimo livello e classi. Python combina una eccezionale potenza con una sintassi estremamente chiara. Ha interfacce verso molte chiamate di sistema, oltre che verso diversi ambienti grafici, ed è estendibile in C e in C++. È inoltre usabile come linguaggio di configurazione e di estensione per le applicazioni che richiedono un'interfaccia programmabile. Da ultimo, Python è portabile: può girare su molte varianti di UNIX, sul Mac, sui PC con MS-DOS, Windows, Windows NT e OS/2". In pratica l'utilizzo di questo linguaggio semplifica certe operazioni altrimenti onerose in linguaggi di basso livello (come *bash*) pur mantenendo lo stesso livello di efficienza e senza cozzare contro i requisiti di portabilità del progetto. A sostegno della scelta effettuata si riporta un esempio che sarà utilizzato spessissimo all'interno dell'applicazione in esame.

```
>>>import os
>>>cmd = 'echo "hello world!"'
>>>os.system(cmd)
hello world!
```

L'esempio vede come protagonista la funzione system che, importata dalla libreria os, permette di eseguire codice *bash* o in generale qualunque programma in qualunque momento all'interno di uno script Python, esattamente come si farebbe tramite una shell .

3.3.2 *Lo script Interrogatore*

In questa sottosezione verrà descritto il fulcro del lavoro svolto che è consistito nello sviluppo di tre serie di script ognuna adibita alla risoluzione di uno dei sottoproblemi sopra citati. La prima serie è stata denominata Interrogatore ed è composta da uno script in bash *st.sh* e da uno script in Python *parsing.py*. Si riportano di seguito in ordine di esecuzione le azioni svolte dal programma:

- Interrogazione alla base di dati della *web application* al fine di estrarre le query da somministrare al motore di ricerca — *Competenza di st.sh*
- Conversione della query i-esima in un formato comprensibie da *Google Search*
- Richiamo dello script *parsing.py*
- Inserimento degli URL restituiti nella base di dati

Competenza di parsing.py

- Apertura della pagina indice relativa alla query ricevuta in input
- Parsing della pagina indice ed estrazione degli URL di interesse
- Passaggio alla pagina indice successiva fino al raggiungimento di un numero di URL soddisfacente (circa 100000)
- Restituzione dei risultati a *st.sh*

Tra i due script la comunicazione avviene attraverso semplici file di testo utilizzati ovviamente in mutua esclusione. Nel caso particolare sono stati creati i file *in.txt* per la comunicazione (*st.sh* in scrittura -> *parsing.py* in lettura) e *output.log* per la comunicazione inversa (*parsing.py* in scrittura -> *st.sh* in lettura). Al termine delle operazioni sia *in.txt* che *output.log* vengono prontamente eliminati da *st.sh*.

3.3.3 *Problemi riscontrati*

I problemi principali qui riscontrati sono stati sostanzialmente due:

1. Il primo riguarda le politiche di Google. Come spesso accade in un motore di ricerca l'automatizzazione, i bot e gli script sono di default bloccati quando tentano di accedere ad una pagina indice.
2. Il secondo, che si manifesta non appena il primo viene risolto, consiste nel fatto che tutte le azioni svolte dagli utenti di Google vengono monitorate. Troppe azioni in troppo poco tempo comportano il ban[2] dal servizio. Lo script Interrogatore cade purtroppo in questa trappola dovendo aprire dalle 1000 alle 10000 pagine indice nel giro di decine di secondi.

Il primo problema è stato risolto abbastanza agevolmente attraverso lo *user-agent identification*. Questo oggetto non è altro che una stringa che permette al sito interpellato (Google.it) di identificare il programma che effettua le richieste (Interrogatore). Nel caso di Interrogatore si è utilizzato lo *user-agent id* di Firefox 2.0, in altre parole lo script agirà come fosse un browser. Tutto ciò è stato implementato grazie ad una funzionalità nativa di Python e si riportano qui le righe di codice significative:

2 Sinonimi di ban sono censura, interdizione, proibizione. Nel caso in esame significa la sospensione del servizio, Google smette di fornire risultati.

```
>>>import urllib
>>>from urllib import FancyURLopener
>>>class MyOpener(FancyURLopener):
      version = 'Mozilla/5.0 (Windows; U; Windows NT 5.1; it;
rv:1.8.1.11) Gecko/20071127 Firefox/2.0.0.11'
>>>myopener = MyOpener()
>>>page = myopener.open(pagina_indice)
>>>p = page.readlines()
```

Lo *user-agent id* è evidenziato in grassetto; attraverso la classe *MyOpener* non si fa altro che creare l'oggetto *myopener* che sarà utilizzato per aprire la pagina indice.

Il secondo problema è invece più delicato, non esiste una particolare funzione che permette di evitare il ban di Google, nessuno sa quali sono le sue strategie di monitoraggio. Tuttavia esistono dei metodi empirici che permettono di rendere meno sospetto un qualunque script innocuo come Interrogatore, lo scopo di questi metodi non è altro che normalizzare la frequenza di accessi dello script in modo da simulare il più possibile la frequenza di visite che farebbe un ipotetico utente umano. Questo si realizza inserendo pause tra una richiesta e l'altra secondo un determinato criterio. Naturalmente di criteri ne esistono diversi ma tra tutti i possibili è stato scelto quello rappresentato in Figura 4.

Ciò che contraddistingue un bot da un essere umano è la prevedibilità. Un programma come Interrogatore non fa altro che eseguire m volte la stessa serie di operazioni ed impiega pressoché lo stesso tempo per ogni iterazione. La funzione anti-ban cerca di inserire delle discontinuità in queste tempistiche in modo da aprossimare il comportamento di un utente umano che di norma effettua visite più discontinue.

Nell'ordine, la prima parte della funzione genera un numero intero, compreso tra 1 e 100 secondo una certa distribuzione di probabilità. Per semplicità è stata adottata una probabilità di tipo uniforme, si è generato un numero casuale q con estremi 1 - 10 e si è effettuata la seguente assegnazione: $P = 10 \cdot q$. Successivamente, con probabilità $\frac{1}{2}$ lo script provvederà ad aspettare un numero casuale ma non eccessivo di secondi prima di procedere con le sue faccende. Come risultato ci si aspetta che metà delle volte lo script attenda qualche tempo prima di riprendere a lavorare. Nonostante la relativa semplicità dell'approccio i test hanno sempre dato (fin'ora) esito positivo ma nel caso si presentasse una futura necessità di modifica sarà opportuno intervenire sulla distribuzione di probabilità oppure sulla durata dell'attesa di ogni iterazione.

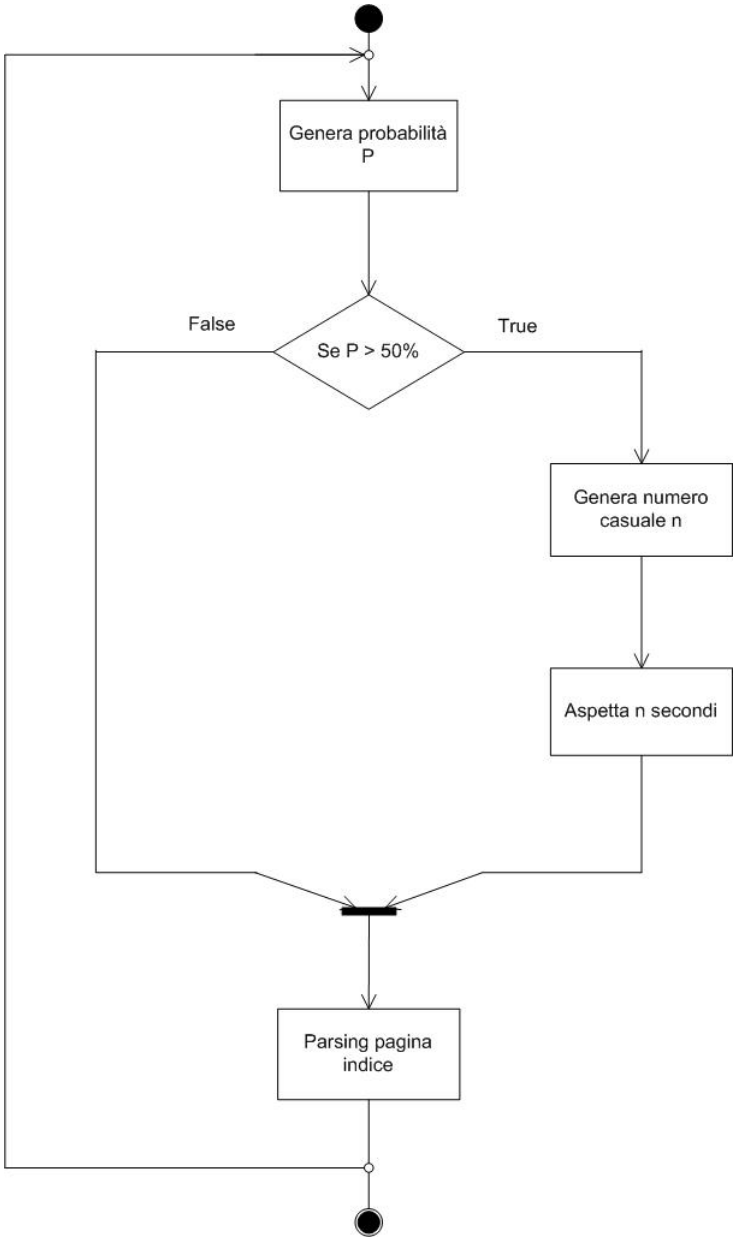

Figura 4: Diagramma attività della funzione anti-ban

3.3.4 *Lo stile degli script: gli input*

Come discusso nelle sezioni precedenti lo script *st.sh* si preoccupa di fornire tutti gli input necessari all'interrogatore e dell'elaborazione dei relativi output. Fondamentalmente *st.sh* non fa altro che interrogare la base di dati al fine di ricavare:

1. il numero di query da dare in pasto al motore di ricerca

2. il testo delle query stesse

Entrambe le operazioni sono implementatate con una riga di codice comprendente alcuni comandi posti in *piping*. Le pipe non fanno altro che interrogare la base di dati per poi trasformare l'output in un risultato più amichevole. Per chiarire le idee di seguito viene presentata la particolare pipe che costruirà l'input per lo script Interrogatore:

```
#estrae le query da risolvere
echo "select query from interrogazione" | psql | tail -n +3 |
head -n -2   cut -d  \  -f2-10> ris
```

Figura 5: Query: "cut and paste"

La prima coppia di comandi scrive l'interrogazione e la passa alla base di dati:

```
marcatof@dbstud:~$echo "select query from interrogazione" | psql
       query
--------------------
 unipd
 nasa
 medicina generale
 "formaggio asiago"
 site:.it
(5 rows)

marcatof@dbstud:~$
```

dall'output ottenuto vengono rimosse le prime due righe...

```
marcatof@dbstud:~$echo "select query from interrogazione" | psql | tail -n +3
 unipd
 nasa
 medicina generale
 "formaggio asiago"
 site:.it
(5 rows)

marcatof@dbstud:~$
```

...e l'ultima

```
marcatof@dbstud:~$echo "select query from interrogazione" | psql | tail -n +3 |
head -n -2
 unipd
 nasa
 medicina generale
 "formaggio asiago"
 site:.it
marcatof@dbstud:~$
```

infine, visto che *parsing.py* può comunicare con *st.sh* via file il tutto viene salvato, appunto, in un file di testo:

```
marcatof@dbstud:~$echo "select query from interrogazione" | psql | tail -n +3 |
head -n -2 | cut -d\   -f2-10 > ris
marcatof@dbstud:~$cat ris
unipd
nasa
medicina generale
"formaggio asiago"
site:.it
marcatof@dbstud:~$
```

In realtà è necessario ancora un accorgimento prima di somministrare le query allo script, Google infatti costruisce l'url delle pagine indice sostituendo ogni spazio con il carattere '+' ed inoltre è necessario sostituire il carattere ' " ' con il rispettivo codice ASCII. Questo viene fatto agevolmente sempre in *st.sh* con le seguenti righe di codice:

```
for ((i=1; i<=$fine; i++))
do
...
echo | sed -n "$i"'p' ris | sed s/\ /\+/g |
sed s/\\"/\\%22/g > in.txt
...
done
```

Dove:

1. $fine è il numero di query (5 nell'esempio),
2. *sed* è il comando che permette di selezionare la riga di interesse per poi manipolarla secondo le nostre esigenze, nel nostro caso, sostituendo a tutti i caratteri <spazio> il carattere '+' e a tutti i caratteri ' " ' la stringa '%22'
3. *in.txt* è il file definitivo che sarà utilizzato da *parsing.py* come input.

3.3.5 *Lo stile degli script: il parsing*

Il cuore del modulo Interrogatore risiede in *parsing.py*, uno script in linguaggio Python che data una stringa contenente la query desiderata esegue un'interrogazione al motore di ricerca per poi restituire in ordine i primi 1000 risultati relativi a tale query. Alcuni interventi di questo script sono già stati svelati nelle sezioni precedenti tuttavia qui si approfondiranno alcuni dettagli di realizzazione.

La scelta dei 1000 URL è riconducibile alla bontà del motore di ricerca: data una qualunque interrogazione Google search seleziona le prime centinaia di risultati e li propone all'utente. È possibile però migliorare la ricerca aumentando il numero di risultati per pagina ed estendendo il numero di risultati restituiti complessivamente. Per applicare queste migliorie è sufficiente modificare l'URL delle pagine indice in maniera opportuna, come nel seguente esempio:

http://www.google.it/search?q=unipd&filter=0&num=100.

1. l'opzione *filter* comunica a Google di fornire il maggior numero di risultati possibile (cioè 1000)
2. l'opzione *num* rappresenta il numero di risultati utili per ogni pagina, il valore assegnabile massimo è 100.

Successivamente va implementata la funzione anti-ban descritta nella sezione precedente. Le funzionalità che servono a Python sono due: libreria time e libreria random.

```
import time
import random
import sys

...
#10 è il numero di pagine indice
for i in range(10):

            #aspetto 3 secondi con probabilita' 1/2
            wait=random.randrange(10)
            if wait > 5:
                  saveout= sys.stdout
                  sys.stdout = sys.__stdout__
                  print 'aspetto 3 secondi...'
                  sys.stdout = saveout
                  time.sleep(3)
...
```

La funzione anti-ban è applicata ad ogni pagina indice, ecco il perchè del suo inserimento nel ciclo *for*. Le varie assegnazioni all'oggetto *sys.stdout* permettono di visualizzare il messaggio "aspetto 3 secondi..." a video anzichè scriverlo su qualunque output precedente (cioè su file).

Una volta che tutti gli accorgimenti preliminari sono stati adottati si richiama il *parser*. La pagina indice, aperta come file di testo, viene analizzata grazie ad un pattern[3] scelto. La funzione di parsing non fa altro che ricercare il pattern riga per riga e una volta trovato provvede ad estrarre l'URL contenuto nel tag HTML corrispondente. Tutte queste operazioni, a livello di codice, si traducono in una sorta di *cut and paste* di stringhe:

3 Per quanto riguarda Google search il pattern scelto è il seguente: '<li class=g'. Questo particolare tag HTML contiene tutti gli URL restituiti in tutte le pagine indice.

```
#salvo la pagina come file di testo
page = myopener.open(queryurl)
p = page.readlines()
#estraggo gli url
for line in p:
     inizio = line.find('<li class=g')
     while (inizio != -1):
          line = line[inizio:]
          inizio = line.find('http')
          line = line[inizio:]
          fine = line.find('"')

          #ecco l'URL estratto:
          url = line[:fine]
          #ecco l'output, a seconda delle opzioni
          #potrà direzionarsi a video, su file o nella
          #base di dati:
          print(url)

          line = line[fine:]
          inizio = line.find('<li class=g')
page.close()
```

Tutto ciò che rimane da gestire a questo punto è l'output del *parser* che, come prima enunciato, è affidato a *st.sh*.

3.3.6 *Lo stile degli script: gli output*

Una volta che lo script in Python avrà eseguito le sue mansioni, a seconda dell'opzione scelta nel menu principale, i risultati verranno visualizzati a video, scritti su file, o memorizzati direttamente nella base di dati. Mentre le prime due hanno scopi di puro debugging l'ultima di queste tre opzioni rende Interrogatore davvero utile.

```
marcatof@dbstud:~/Tesi Anno III/Applicazione$python main.py

 ------> MENU <------

Ricava URLs 1)
Campiona 2)
Converti per la visione Offline 3)
Esci 4)
 1
numero interrogazioni: 5

Visualizza a video 1)
Inserisci nel db 2)
Scrivi su file 3)
Annulla questa interrogazione: unipd 4)
Torna al menu principale 5)
```

Figura 6: Menu principale

```
Visualizza a video 1)
Inserisci nel db 2)
Scrivi su file 3)
Annulla questa interrogazione: unipd 4)
Torna al menu principale 5)
 1


----------------- RISULTATO PAGINA NUMERO 1

url query: http://www.google.it/search?q=unipd&filter=0&num=10

1http://www.unipd.it/
2http://www.unipd.it/strutture/facolta/facolta.htm
3http://www.cca.unipd.it/posta/
4http://www.lettere.unipd.it/
5http://www.unipd.it/offerta_didattica/index.htm
6http://www.unipd.it/offerta_didattica/
7http://www.unipd.it/servizionline/
8http://www.psy.unipd.it/
9http://www.unipd.it/dirittoallostudio/
10http://www.ing.unipd.it/
11http://www.medicina.unipd.it/
12http://www.scipol.unipd.it/
13http://www.giuri.unipd.it/
14http://www.formazione.unipd.it/
15http://www.psicologia.unipd.it/
16http://www.unipd.it/area/area-4.htm
```

Figura 7: Menu parser: Visualizza a video

La visualizzazione permette di identificare il numero della pagina indice, l'URL della query somministrata a Google Search e tutti i risultati numerati progressivamente.

Con l'opzione "Scrivi su file" otteniamo un risultato forse meno leggibile ma sicuramente più vicino a ciò che realmente verrà memorizzato nella base di dati:

```
Visualizza a video 1)
Inserisci nel db 2)
Scrivi su file 3)
Annulla questa interrogazione: unipd 4)
Torna al menu principale 5)
 3

Visualizza a video 1)
Inserisci nel db 2)
Scrivi su file 3)
Annulla questa interrogazione: nasa 4)
Torna al menu principale 5)
 5

 ------> MENU <------

Ricava URLs 1)
Campiona 2)
Converti per la visione Offline 3)
Esci 4)
 4
marcatof@dbstud:~/Tesi Anno III/Applicazione$ls
main.py  pagine.txt  parser  sampler  savewpage  webdb
```

Figura 8: Menu parser: Scrivi su file

Estratto di pagine.txt

```
marcatof@dbstud:~/Tesi Anno III/Applicazione$cat pagine.txt | head -n +20
http://www.unipd.it/
http://www.unipd.it/strutture/facolta/facolta.htm
http://www.cca.unipd.it/posta/
http://www.lettere.unipd.it/
http://www.unipd.it/offerta_didattica/index.htm
http://www.unipd.it/offerta_didattica/
http://www.unipd.it/servizionline/
http://www.psy.unipd.it/
http://www.unipd.it/dirittoallostudio/
http://www.ing.unipd.it/
http://www.medicina.unipd.it/
http://www.scipol.unipd.it/
http://www.giuri.unipd.it/
http://www.formazione.unipd.it/
http://www.psicologia.unipd.it/
http://www.unipd.it/area/area-4.htm
http://www.unipd.it/sis/
http://www.math.unipd.it/
http://www.scform.unipd.it/
http://www.dei.unipd.it/
marcatof@dbstud:~/Tesi Anno III/Applicazione$
```

L'opzione "Visualizza a video" è l'ideale per riconoscere e controllare tutti i parametri che costituiscono la comunicazione tra *st.sh* e *parsing.py*. La scelta di

inserimento nella base di dati produce invece una serie di messaggi a video che testimoniano le operazioni di cancellazione o inserimento di Interrogatore. *St.sh* è in grado infatti sia di popolare la base di dati che di aggiornarla, interrogazione per interrogazione, a seconda delle esigenze. Le azioni che effettivamente vengono svolte a questo riguardo sono:

1. Eliminazione dei vecchi url dalle tre tabelle URL, RESTITUZIONE e RILEVANZA.

2. Inserimento nelle medesime tabelle degli url aggiornati.

A questo punto Interrogatore ha terminato il suo compito, la base di dati è popolata da migliaia di URL grezzi, pronti per essere campionati e messi a disposizione della *web application*: è il turno dello script Campionatore.

3.4 FUNZIONALITÀ II: CAMPIONAMENTO DEGLI URL

Il lavoro di Interrogatore ha prodotto una popolazione di URL soddisfacente dal punto di vista qualitativo ma il numero di risultati memorizzati nella base di dati, rapportato con il numero di utenti finali stimati, risulta decisamente eccessivo. Il lavoro della funzione di campionamento potrebbe essere paragonata alla stesura di un riassunto: quando uno studente deve elaborare una tesi è importante che raccolga quante più informazioni possibile ma deve assicurarsi di mantenere nello stesso tempo un volume di pagine adeguato per non rendere pesante la lettura al suo relatore. In quattro parole SSPI, sintesi senza perdere informazione. La popolazione di URL ha bisogno di essere raffinata e sfoltita ma non deve perdere di significato. In questa sezione si analizzerà lo script detto Campionatore che si preoccuperà di eliminare gli URL di troppo secondo un principio euristico approvato.

3.4.1 *La scelta*

Campionatore è stato progettato per essere interattivo. Lo script infatti permette all'utente di scegliere una delle teorie di campionamento previste dal progettista. Si noti tuttavia che nella versione 1.0 sarà inserita una sola teoria. Nulla però vieterà di inserirne di nuove in successive *release*. In questa relazione non ci si dilungherà sugli aspetti prettamente matematici della scelta tuttavia si fisseranno le idee sugli aspetti principali che la caratterizzano.

Nella distribuzione di rilevanza degli URL abbiamo un'alta concentrazione di URL rilevanti a ridosso delle prime pagine indice e URL sempre meno rilevanti man mano che si prendono in considerazione pagine indice successive. Il lavoro di Campionatore consisterà in una serie di cancellazioni di tuple al fine di massimizzare richiamo e precisione. La prima rappresenta quante pagine rilevanti

sono state catturate tra tutte quelle presenti nella popolazione, la seconda rappresenta quante delle pagine che abbiamo catturato sono rilevanti. Sfortunatamente queste due misure sono inversamente proporzionali tra di loro ed è quindi necessario trovare un compromesso accettabile. La soluzione detta "campionamento a grappoli" cerca di assicurare proprio questo. In particolare questa metodologia consiste in un campionamento sistematico della popolazione con passo k (nel caso in esame, la popolazione di riferimento è la popolazione degli URL nella base di dati).

3.4.2 *Lo script Campionatore*

La seconda serie di script è denominata Campionatore così come si vede in Figura 2. La composizione di questo programma è semplicissima: uno script in Python chiamato *sampler.py*.

Al suo interno *sampler.py* è dotato di un menu autosufficiente che, una volta attivato, provvede a lanciare il campionamento su tutte le tabelle coinvolte. L'utente che abbia intenzione di utilizzare questa funzionalità non dovrà far altro che:

1. Selezionare lo stile di campionamento che soddisfi le sue esigenze
2. Specificare il passo di campionamento per ogni grappolo
3. Specificare la grandezza di ogni grappolo
4. Aspettare la fine delle elaborazioni

Tutte le operazioni sono svolte direttamente sulla base di dati pertanto non sono stati previsti output alternativi.

3.4.3 *Problemi riscontrati*

Non sono stati riscontrati particolari problemi nell'implementazione di questa funzionalità. Durante la fase di testing si sono apportate piccole modifiche all'interfaccia utente migliorando l'usabilità del programma in particolare per i punti 2 e 3 della sottosezione precedente.

3.4.4 *Lo stile dello script: gli input*

In questa sottosezione si analizzeranno più da vicino alcune scelte implementative adottate in *sampler.py*. Per quanto riguarda il campionamento "a grappoli" sono state selezionate tre classi di campioni. È importante notare che il campionamento a grappoli effettua sostanzialmente le stesse operazioni per tutte e tre le classi, solo con passo diverso. A fronte di questo è stata implementata una funzione detta *cluster_sampling* che dati come parametri il numero di query, il passo e la

classe di campionamento, effettua cancellazioni sistematiche secondo il principio di campionamento stesso. Per quanto riguarda parametro "numero di query" l'idea è quella di interrogare la base di dati, trovare quanti URL sono realmente stati memorizzati e quindi ricavare il numero di interrogazioni con una semplice divisione:

```
import os
#commands permette di salvare l'output di un comando
import commands

#determina quanti url sono stati finora memorizzati
#nella base di dati
cmd = 'echo "select max(idurl) from url" | psql | tail -n +3 |
head -n -2 | cut -d\  -f2'
nurl = int(commands.getoutput(cmd))
nquery = nurl / 100000
```

Merita particolare attenzione la libreria *commands* che in maniera simile alla sorella *os* permette di eseguire codice *bash* senza uscire dallo script. La scelta della funzione *commands.getoutput(cmd)* è dovuta al fatto che questa permette di catturare eventuali valori restituiti dai comandi passati come parametro. Nel caso in esame è esattamente ciò che succede, con la solita strategia *cut and paste* la base di dati viene interrogata e questa prontamente risponde con una stringa rappresentante il valore desiderato.

All'interno del corpo dello script si possono individuare tre distinte chiamate a *cluster_sampling*. I rispettivi parametri sono valorizzati secondo le preferenze dell'utente quindi avremo:

- Grappolo 1
  - viene posto l'indirizzo[4] iniziale a zero
  - gli altri paramentri vengono calcolati dinamicamente
- Grappolo 2
  - l'indirizzo iniziale è coincidente all'indirizzo finale del grappolo precedente
  - gli altri paramentri vengono calcolati dinamicamente
- Grappolo 3
  - l'indirizzo iniziale è coincidente all'indirizzo finale del grappolo precedente
  - viene posto l'indirizzo finale a 1000
  - il passo viene calcolato dinamicamente

4 La chiave primaria di tutte le tabelle è memorizzata con il seguente formato: <10000*numero interrogazione>+<numero URL>. L'indice è esattamente il "numero dell'URL".

3.4.5 *Lo stile dello script: il campionamento*

La funzione *cluster_sampling* viene invocata tre volte all'interno dello script, una per ogni grappolo. Ad ogni invocazione viene eseguita una serie di operazioni relativamente semplice, in particolare:

1. Si determina la chiave primaria del primo URL da sottoporre al campionamento
2. Si eseguono le cancellazioni mirate alle tre tabelle URL, RESTITUZIONE e RILEVANZA
3. Si comunica l'operazione all'utente

Ci sono due aspetti che vale la pena approfondire, il primo di questi riguarda la gestione dei primi k URL: se si applicassero le rimozioni sistematiche in maniera meccanica i primi k URL rimarrebbero sempre nella popolazione campione finale. Per attenuare questa ingiustizia ogni qual volta la funzione viene richiamata si incrementa l'indice del primo URL di un valore casuale tra o e k dove k è proprio il passo di campionamento; in questa maniera ad ogni invocazione la parte iniziale della popolazione campione sarà sempre differente.

```
#Campionamento a grappoli di passo k
#@params nquery: numero di interrogazioni da campionare
#@params k: passo di campionamento
#@params start: indice del primo url da campionare
#@params end: indice dell'ultimo url da campionare
def cluster_sampling(nquery, k, start, end):
...
    begin = random.randrange(k)
    print "\nNumero speciale: %s\n" % begin
    time.sleep(2)
    j = begin + start
...
```

Una seconda questione riguarda la gestione del parametro "passo di campionamento". A seconda del valore del passo la funzione deve campionare più o meno frequentemente, l'idea è quella di utilizzare l'operatore *mod* tra la variabile j e il passo k, ecco come (in grassetto):

```
def cluster_sampling(nquery, k, start, end):
...
    while( j < end ):
        if(( j - begin ) % k != 0 ):
         #i è il numero della query
        cmd='echo "delete from rilevanza where
            url=(' + str((i+1)*100000+j) + ')" | psql \n
        cmd= cmd + 'echo "delete from restituzione where
            url=(' + str((i+1)*100000+ ) + ')" |  s l \n
        cmd= cmd + 'echo "delete from url where
          idurl=(' + str((i+1)*100000+ ) + ')" |  s l \n
        os.system(cmd)
     j = j+1
```

la variabile j rappresenta l'indice, ovvero la parte significativa della chiave primaria degli URL.

Una volta che il campionamento è andato a buon fine la *web application* sarà in grado di funzionare a pieno regime. Ma c'è ancora un vincolo da soddisfare: il web è un ambiente estremamente mutevole; se lo scopo del progetto corrente è quello di raccogliere delle opinioni sulla qualità delle pagine allora è necessario che tutti gli utenti futuri votino, chiaramente, sulla stessa popolazione. Così come stanno le cose dall'oggi al domani è più che possibile che un sito, anzi diversi siti cambino la propria *skin*, oppure la lingua, i form o potrebbero addirittura chiudere! È il momento di introdurre l'ultima coppia di script: Downloader.

3.5 FUNZIONALITÀ III: DOWNLOAD DELLE PAGINE WEB

Se si volesse dividere in gruppi le funzionalità della *web application* allora se ne dovrebbero individuare due: il primo, quello delle funzionalità critiche, conterrebbe Interrogatore e Campionatore; il secondo, invece, conterrebbe questa terza componente: il Downloader.

Il problema esposto nelle ultime righe della sezione precedente non è così banale: è necessario mantenere la popolazione omogenea per una durata di tempo sufficiente altrimenti utenti diversi potrebbero ridursi a valutare uno stesso URL ma pagine completamente diverse, falsando così la valutazione finale. La componente che verrà qui esposta ha il compito di prelevare gli URL forniti dalla base di dati e di fungere da *offline browser*, si preoccuperà cioè di fotografare le pagine web in un determinato momento per poi ritornarle all'applicazione principale.

3.5.1 *La scelta*

Come nel caso di Interrogatore e prima ancora nel caso dell'intero software di campionamento ci si è trovati a dover scegliere la strategia di implementazione più conveniente. Storicamente le vie prese in considerazione sono state tre:

1. Screenshot delle pagine web mediante il tool webkit2png di Python
2. Download completo delle pagine attraverso il comando *wget* di *bash*
3. Download completo delle pagine attraverso un software di terze parti come httrack

In prima analisi la soluzione numero uno risultava facilmente implementabile ma la semplicità ha presto comportato anche incompletezza e incompatibilità rispetto agli obiettivi che ci si era posti a monte. L'idea faceva leva su aspetti positivi come consumo di memoria accettabile, velocità di esecuzione, indipendenza dalla natura della pagina stessa; tuttavia altri aspetti come scarsa portabilità e incapacità di catturare animazioni in formato *flash* hanno comportato l'abbandono di tutta l'ipotesi.

Il comando *wget* , tipico del linguaggio *bash*, ha assorbito diverso tempo sia per la valutazione che per il test. La realizzazione è partita con uno sviluppo ex-novo ma purtroppo la complessità del problema e alcune limitazioni del comando stesso hanno portato all'abbandono anche di questa soluzione.

Infine finalmente httrack è intervenuto quando tutte le strade percorribili sembravano essere state percorse. Il software qui nominato è una *free, easy-to-use, offline browser utility* così come è definito sul sito ufficiale [7, `http://python.org/`]. In altre parole httrack è un programma open source che mette a disposizione una serie di utilità per il download di siti web. È doveroso notare che non presenta problemi di portabilità in quanto è compatibile con qualunque sistema operativo (a patto che si scarichi la versione corretta) e la presenza di un numero decisamente alto di opzioni rende questo prodotto duttile a qualunque lavoro si intenda svolgere. Quindi Httrack è stata la soluzione effettivamente adottata in Downloader.

3.5.2 *Lo script Downloader*

Il programma Downloader è formato come Interrogatore da una coppia di script: uno in *bash* chiamato *stsave.sh* e uno in Python chiamato *htpage.py*. Le funzionalità di *stsave* si esauriranno presto in quanto la sua sola utilità è quella di fornire tutti gli URL di interesse dalla base di dati e di richiamare *htpage.py* alle sue mansioni. Ecco quindi quali sono le azioni svolte da Downloader in ordine di esecuzione:

competenza di stsave.sh

- richiamo degli URL dalla base di dati

- visualizzazione di un menù interattivo che permette di scaricare tutti gli URL o solo parte di questi *competenza di htpage.py*
- indentificazione della natura dell'URL
  - se questo è un file allora attiva il download dedicato ai file
  - in alternativa attiva il download standard

Httrack è utilizzabile in modalità grafica oppure come scelto in questa occasione, da riga di comando. Una nota sull'utilizzo di questa modalità è da farsi in quanto l'ambiente di sviluppo messo a disposizione dal dipartimento non permette l'installazione di alcun software di terze parti per questioni di sicurezza. Per rimediare a questo inconveniente senza scomodare i sistemisti si è utilizzata la versione portabile di httrack (ovvero le sue librerie), e si è impostato un comando personalizzato chiamato *ht* che richiamerà le funzionalità desiderate da tale versione.

3.5.3 *Problemi riscontrati*

Una volta che è stata resa possibile l'installabilità di Httrack i problemi più immediati sono stati quelli legati alla natura del software. Come ogni pacchetto open source che si rispetti anche questo ha richiesto una buona dose di addestramento prima del suo utilizzo a pieno regime. Non sono mancati i dubbi sulla sintassi, sul significato dei comandi e sull'effettivo funzionamento. Tuttavia grazie ad un tutorial ben scritto e ad una comunità pronta a rispondere ai problemi più disparati il tempo messo a disposizione è stato più che sufficiente per capire le potenzialità di questo software e di sfruttarle al meglio nel caso di Downloader.

Oltre a tali punti di forza è però necessario mettere in luce anche i colli di bottiglia di Httrack:

1. Velocità di esecuzione
2. Memoria di massa necessaria

Per quanto riguarda il primo punto si è potuto fare davvero ben poco, Httrack legge la pagina, effettua parsing, scarica tutti i file necessari tra immagini, video e allegati, bypassa protezioni, effettua elaborazioni. Tutte queste operazioni dipendono dalla banda, dalla macchina ma anche dal software stesso che pare prendersi almeno qualche minuto per un sito complesso come `http://www.unipd.it`. Vien da sè che per la pagina e tutti i file necessari alla sua visualizzazione è necessario mettere a disposizione più memoria di quella che si avrebbe bisogno per uno snapshot .png.

3.5.4 *Lo stile degli script: gli input*

Come accennato nella sezione generale l'input implicito di Downloader è esattamente l'output di Campionatore. Una volta che gli URL sono pronti nella base di dati questi vengono prelevati da *stsave.sh* e con la solita tecnica *cut and paste* scritti su file. *htpage.py* oltre a questo ha bisogno della preferenza dell'utente per capire cosa fare. È presente un menu a quattro scelte che permettono, nell'ordine, di:

1. Scaricare tutte le pagine presenti nella base di dati
2. Scaricare la prima pagina corrispondente al primo URL
3. Scaricare le prime n pagine con n numero inserito da riga di comando
4. Tornare al menu principale

Come per Interrogatore si possono individuare due classi di opzioni: opzioni riservate alla versione stabile (la numero uno) e opzioni riservate al testing (la numero due e tre).

```
marcatof@dbstud:~/Tesi Anno III/Applicazione$python main.py

 ------> MENU <------

Ricava URLs 1)
Campiona 2)
Converti per la visione Offline 3)
Esci 4)
 3

 ------> MENU: scegli il numero di pagine da scaricare <------

TUTTE (1)
Una (2)
Custom (3)
Indietro al menu' principale (4)
```

Figura 9: Menu Downloader

3.5.5 *Lo stile degli script: il download*

A seconda della scelta nel menu sopracitato Httrack si attiverà scaricando gli URL selezionati e sotto supervisione di *Htpage.py* imposterà il tipo di download più appropriato. Quest'ultima selezione è implementata in un classico blocco *if* dove si controlla l'estensione dell'URL stesso: se questa non fa parte di una famiglia comune come .htm, .php, .shtml allora il download viene eseguito con le seguenti opzioni:

```
cmd='ht -I -g -O myweb/'+url+' "'+url+'"'
```

Nell'ordine:

1. *ht* è il comando personalizzato che invoca httrack in versione linea di comando
2. si comunica di creare un index
3. si specifica di scaricare in modalità file
4. si specifica la directory destinazione con percorso myweb/nomeURL
5. si segnala infine l'URL da scaricare

Se il risultato del test segnala che la pagina ha estensione comune vengono attivate opzioni sensibilmente diverse:

```
cmd='ht -r 2 -n -%P -s0 -F '+useragent+' -x -G10000000
     -m500000 -O myweb/'+dest+' "'+url+'"'
```

Nell'ordine:

1. Si invoca httrack
2. si richiede una profondità di download massima pari a due, questa scelta permette di scaricare tutti i file necessari alla visualizzazione
3. si comunica ad httrack di interpretare il JavaScript
4. si ignora il file robot.txt che solitamente contiene restrizioni per questo genere di operazioni
5. si specifica uno *user agent identification*
6. si specifica di fare una pausa dopo 10MB, simula la funzione anti-ban di Interrogatore
7. si specifica un *upper bound* alla grandezza di ogni file html pari a 200KB
8. si specifica un *upper bound* alla grandezza di ogni file non html pari a 5MB
9. si specifica la directory di destinazione in maniera analoga al caso precedente
10. si segnala infine l'URL da scaricare

3.5.6 *Lo stile degli script: gli output*

Una volta specificate tutte le opzioni, detti filtri in ambiente Httrack, Downloader procederà ad eseguire le operazioni richieste costruendo un albero di cartelle e sottocartelle del tipo myweb/nomeURL. Può capitare talvolta che URL diversi rientrino in realtà nello stesso dominio, pertanto si è scelto di mantenere la struttura nativa dei siti costruendo un percorso simile anche su disco. Supponendo che vengano dati in pasto a Downloader due URL come `www.archlinux.it/` e `www.archlinux.it/forum` allora il percorso finale costruito dall'applicazione risulterebbe il seguente: myweb/www.archlinux.it/ per il primo e myweb/www.archlinux.it/forum per il secondo. Tali percorsi verranno poi scritti sulla base di dati nell'apposito campo della tabella URL in modo da essere subito disponibili per la *web application*.

CONCLUSIONI

Attualmente non esiste uno studio sperimentale completo che assicuri una corrispondenza tra i punteggi forniti dagli algoritmi di *link analysis* e l'effettiva qualità delle pagine web corrispondenti. In questa relazione è stata descritta un'applicazione di supporto per la verifica dell'efficacia di tali algoritmi. Il software prodotto è costituito da diverse componenti come la base di dati, l'interfaccia web, le funzionalità di campionamento e il sistema di votazioni. Per quanto riguarda le funzionalità di campionamento sono state sviluppate tre serie di script denominate Interrogatore, Campionatore e Downloader per soddisfare le seguenti esigenze:

1. Recupero URL dal motore di ricerca
2. Campionamento pagine secondo la teoria "a grappoli"
3. Download delle pagine su disco

Per garantire il rispetto dei vincoli di portabilità e usabilità ogni script è stato progettato per essere il più possibile interattivo e indipendente sia dagli altri script, sia dall'applicazione nel suo complesso. Come linguaggio di programmazione è stato utilizzato [6, Python] che è rilasciato in diverse versioni nel complesso compatibili con qualunque sistema operativo. I test hanno confermato che il software opera in tempi ragionevoli e che è ben predisposto ad eventuali integrazioni con teorie future di campionamento.

A

DOCUMENTAZIONE BASI DI DATI

A.1 REQUISITI STRUTTURALI

La base di dati che verrà di seguito descritta atterrà al compito di memorizzare una serie di dati per quanto riguarda utenti, algoritmi di *link analysis*, interrogazioni e soglie di supporto per l'applicazione web.

Frasi per Interrogazione

L'utente deve avere l'opportunità di scegliere su quale interrogazione effettuare la procedura di raccolta. Per ogni interrogazione viene messo a disposizione il testo dell'interrogazione, una breve descrizione sull'argomento trattato e un contatore che misura il numero di pagine assegnate o già valutate dagli utenti relativamente alla *query* in esame. Inoltre per ottimizzare le operazioni previste, viene memorizzata anche l'interrogazione corrente per ogni utente, il numero massimo di *query* assegnabili e il numero di *query* effettivamente assegnate per ogni URL. È predisposto infine, un ID che identifica univocamente l'interrogazione stessa.

Frasi per Descrizione Compiti

Bisogna informare l'utente su quali siano i suoi compiti e su come si svolgerà la valutazione. Verrà memorizzato quindi un breve tutorial in formato testuale.

Frasi per Soglia

L'applicazione web necessita di una serie di parametri per quanto riguarda la gestione.

Frasi per Restituzione

Per il voto dell'utente si vuole memorizzare un primo indice chiamato Rilevanza.

Operazione	Tipo
Visualizzazione tutorial progetto	interrogazione
Iscrizione utente	modifica
Modifica credenziali utente	modifica
Recupero password	interrogazione
Scelta dell'interrogazione	interrogazione
Aggiornamento voti	modifica
Visualizzazione interrogazioni richiamate fino ad oggi	interrogazione

Tabella 1: Operazioni sui dati

Frasi per Parametro di Qualità

Per ogni parametro di qualità è di interesse memorizzare il nome, una descrizione e un ID che lo identifica univocamente. Ha senso esprimere un voto per il parametro di qualità se la rilevanza relativa è maggiore o uguale a 6.

Frasi per Indirizzo Web (URL)

Per quanto riguarda la pagina in questione è di interesse memorizzare l'indirizzo web, il percorso su disco (se presente) e un ID numerico che lo identifica univocamente.

Frasi per Algoritmo di Ricerca

Per ogni algoritmo di ricerca rappresentiamo un nome, una descrizione testuale e un ID alfa-numerico che lo identifica univocamente.

Frasi per QualityScore

Ogni algoritmo di ricerca emette un punteggio per ogni URL o interrogazione e si assume che possa valutare più URL e interrogazioni. Perciò per ogni QualityScore rappresentiamo un valore numerico e un ID alfa-numerico che lo identifica univocamente.

A.2 PROGETTAZIONE CONCETTUALE

Vengono descritte nel seguente dizionario tutte le entità e le associazioni di rilievo presenti in Figure 3 .

Entità	Descrizione	Attributi	ID
Utente	utente dell'applicazione	IDUtenti, nickname, password, nome, cognome, sesso, nazionalità, data di nascita, professione, mail, matricola, nummail, lastmail	IDUtenti
Interrogazione	le interrogazioni proposte all'utente	IDInterr, query, descrizione, contatore	IDInterr
URL	la pagina da votare	IDUrl, indirizzo, path	IDUrl
Rilevanza	associa ad ogni pagina un voto di rilevanza	votoRilevanza	IDUtenti, IDInterr, IDUrl
Parametro Qualità	rappresenta uno degli indici di qualità su cui una pagina viene valutata	IDParametro, voto, descrizione	IDParametro
Quality Score	il punteggio restituito da un algoritmo di link analysis sulla pagina	IDScore, valore	IDScore
Algoritmo di Ricerca	uno degli algoritmi di link analysis presi in esame	IDAlg, nome, descrizione	IDAlg
Descrizione Compiti	memorizza un tutorial in formato testuale sui compiti dell'utente	IDDescrizione, testo	IDDescrizione
Soglie	memorizza valori numerici utili per l'applicazione web	IDSoglia, descrizione, valore	IDSoglia

Tabella 2: Dizionario dei dati

Associazione	Attributi	Entità collegate	Descrizione
Selezione	corrente	Utente (1,n) - Interrogazione (0,n)	memorizza l'ultima interrogazione selezionata dall'utente
Restituzione	scelte, maxScelte	Interrogazione (1,n) - URL (1,n)	numero di assegnazioni corrente e numero di assegnazioni massimo (entrambi relativi ad un utente)
Valutazione	voto	Rilevanza (0,n) - Parametro Qualità (0,n)	voto relativo ad una pagina rilevante

Tabella 3: Dizionario Associazioni

A.3 PROGETTAZIONE LOGICA

Lo schema logico-relazionale della base di dati porta ad una struttura come quella illustrata in Figure 10, in cui vengono rappresentate le entità descritte in precedenza sotto forma di tabelle di una base di dati. Inoltre vengono evidenziate le relazioni che intercorrono tra le diverse entità.

Tra UTENTE e INTERROGAZIONE è presente un'associazione molti-a-molti, infatti si vuole dare l'opportunità agli utenti di scegliere più di una interrogazione da analizzare, eventualmente controllando che prima di fornire tale opportunità, sia terminata l'analisi dell'interrogazione precedente. Questa relazione è rappresentata dalla tabella SELEZIONE che unisce, appunto, INTERROGAZIONE e UTENTE.

È possibile che le interrogazioni al motore di ricerca restituiscano una serie di pagine il cui indirizzo si trovi tra i risultati di più *query*. Quindi come indicato dallo schema ER, l'associazione tra INTERROGAZIONE e URL è di tipo molti-a-molti e viene realizzata attraverso la tabella RESTITUZIONE la quale rappresenta ciascuna coppia *query*-risultato.

Le pagine ottenute dalle ricerche, vengono assegnate agli utenti per l'analisi. Tale assegnazione è realizzata con l'inserimento di un dato nella tabella RILEVANZA la quale provvede a memorizzare il voto di rilevanza comunicato dall'utente. Per comodità, la relazione in questione viene indicizzata dal contatore IDRil ed è per tanto opportuno includere un vincolo di non nullità e univocità sul medesimo attributo.

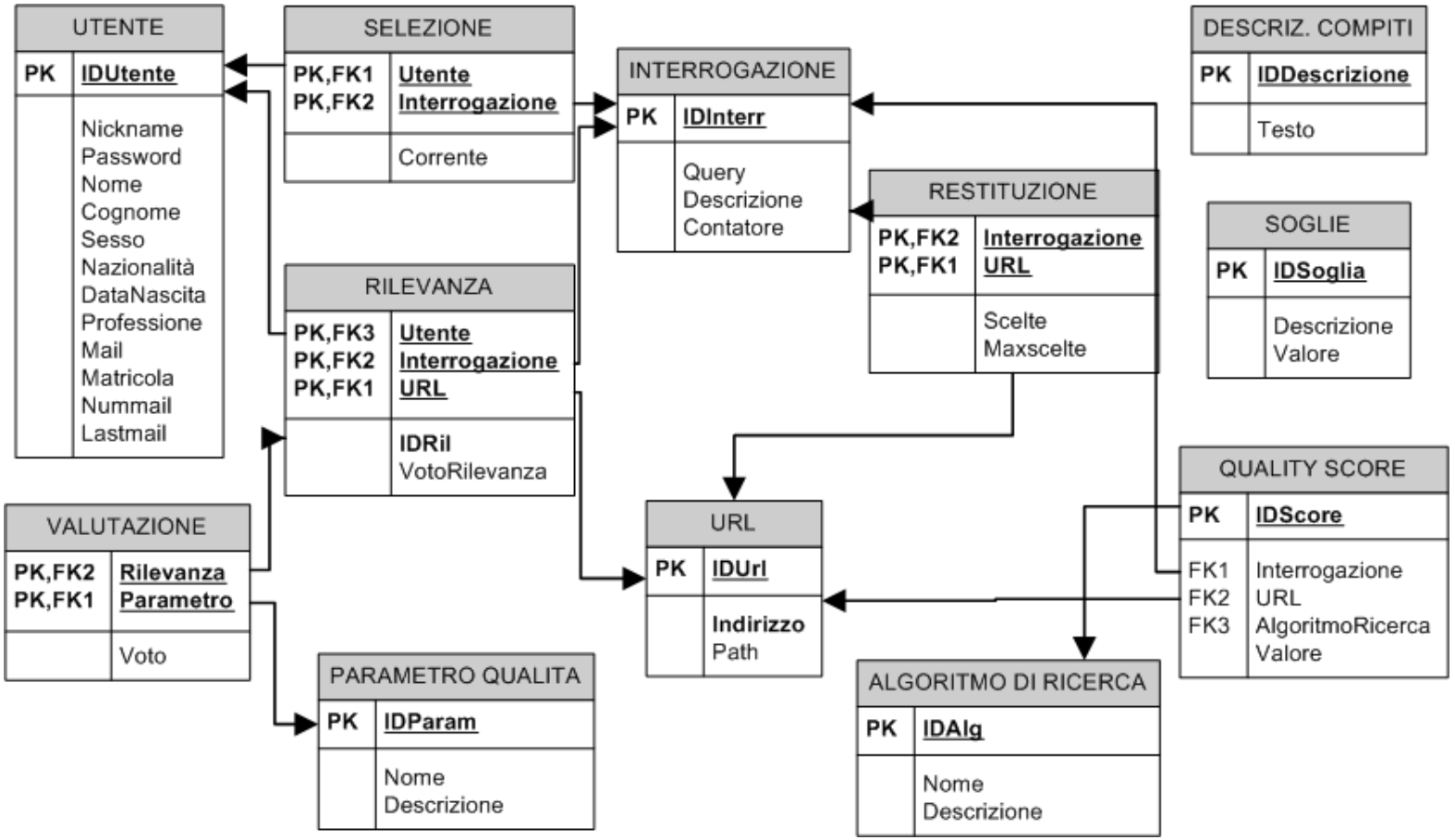

Figura 10: Schema logico

A.4 CODICE SQL

```
#CREATE TYPE genere AS ENUM ('M', 'F');

CREATE TABLE UTENTE (IDUtente integer PRIMARY KEY, Nickname varchar(50),
    Password varchar(20) NOT NULL, Nome varchar(50), Cognome varchar(50),
    Sesso genere, Nazionalita varchar(20), DataNascita date, Professione
    varchar(50), Mail varchar(50), Matricola integer, NumMail smallint
    DEFAULT 0, LastMail date);


CREATE TABLE INTERROGAZIONE (idinterr integer PRIMARY KEY, Query varchar(50)
     NOT NULL, Descrizione text, Contatore integer DEFAULT 0);


CREATE TABLE SELEZIONE (utente integer REFERENCES UTENTE (IDutente),
    interrogazione integer REFERENCES INTERROGAZIONE (idinterr), Corrente
    BOOLEAN DEFAULT FALSE, PRIMARY KEY (utente, interrogazione));


CREATE TABLE URL (IDUrl integer PRIMARY KEY, Indirizzo character varying NOT
     NULL, path character varying);


CREATE TABLE RESTITUZIONE (interrogazione integer REFERENCES INTERROGAZIONE
    (idinterr), Url integer REFERENCES URL (IDUrl), scelte smallint DEFAULT
    0, maxscelte smallint DEFAULT 1, PRIMARY KEY (interrogazione,url));


CREATE TABLE RILEVANZA (utente integer REFERENCES UTENTE (IDutente),
    interrogazione integer REFERENCES INTERROGAZIONE (idinterr), url integer
     REFERENCES URL (IDUrl), idril integer UNIQUE, votorilevanza smallint
    DEFAULT -1, PRIMARY KEY (utente, interrogazione, url));


CREATE TABLE PARAMETROQUALITA (idparam integer PRIMARY KEY, Nome varchar(20)
    , Descrizione text);


CREATE TABLE VALUTAZIONE (rilevanza integer REFERENCES RILEVANZA (idril),
    parametro integer REFERENCES PARAMETROQUALITA (idparam), Voto smallint,
    PRIMARY KEY (rilevanza, parametro));


CREATE TABLE ALGORITMO_DI_RICERCA (IDAlg integer PRIMARY KEY, Nome varchar
    (20), Descrizione text);
```

```
CREATE TABLE QUALITY_SCORE (IDScore integer PRIMARY KEY, interrogazione
    integer REFERENCES INTERROGAZIONE (idinterr), url integer REFERENCES URL
     (IDUrl), AlgoritmoRicerca integer REFERENCES algoritmo_di_ricerca(IDAlg
    ), Valore smallint);


CREATE TABLE DESCRIZCOMPITI (IDDescrizione integer PRIMARY KEY, Testo text);


CREATE TABLE SOGLIE (IDSoglia integer PRIMARY KEY, Descrizione text, Valore
    integer);

GRANT select, update, delete, insert on UTENTE, INTERROGAZIONE, SELEZIONE,
    URL, RESTITUZIONE, RILEVANZA, VALUTAZIONE, QUALITY_SCORE, SOGLIE to
    webdb;
```

BIBLIOGRAFIA

[1] D. Fallows. *The internet and daily life*. 2004. http://www.pewinternet.org. (Citato a pagina 1.)

[2] S. Brin and L. Page. *The anatomy of a large scale hypertextual Web search engine*. In *Proc. of the WWW Conference*, 1998.

[3] J.M. Kleinberg. *Authoritative sources in a hyperlinked environment*. pages 604–632, 1999. Journal of the ACM.

[4] R. Lempel and S. Moran. *SALSA: The stochastic approach for link-structure analysis*. pages 131–160, 2001. ACM Transactions on Information Systems.

[5] *Hitwise, Top 20 Sites and Engines*, Ultima visita: Agosto 2010. http://www.hitwise.com/us/datacenter/main/dashboard-10133.html.

[6] *Python - Official Website*, ultima visita: Agosto 2010. http://python.org/.

[7] *Httrack Website Copier*, ultima visita: Maggio 2010. http://www.httrack.com/.

[8] A. J. Aslam, V. Pavlu, and R. Savell. *A Unified Model for Metasearch, Pooling, and System Evaluation*.

[9] D. C. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge, UP, online edition, 2009.